**Spring AOP (Aspect-Oriented Programming) Notes**

- AOP complements OOP by addressing cross-cutting concerns. ( Normally in web apps, each layer has distinct responsibilities, but certain tasks like validation, logging, security, and caching are common across layers and are known as cross-cutting concerns)



**-** We use this framework to enable middleware services with the spring application in a loosely coupled manner.

- Implementing these cross-cutting concerns in each business logic separately:

* Increase the code size
* Makes code more complex
* Makes the code difficult to manage

AOP separates cross-cutting concerns from business logic **->** Cross-cutting concerns are coded separately **->** Weaving links these concerns with business logic **->** Enhances modularity in the application **->** Simplifies code management and maintenance.

Therefore, AOP separates services from the actual business code.



**Aspect:** Module encapsulating advice and pointcuts.

**Advice:** Code invoked during method execution.

**Pointcuts:** Conditions to execute aspects.

**Join point:** Possible position in business method for applying advice.

**Example 1:**

*import org.aspectj.lang.annotation.\*;*

*import org.springframework.stereotype.\*;*

*@Aspect*

*@Component*

*public class LoggingAspect {*

*@Before("execution(\* Calculator.\*(..))")*

*public void logBefore() {*

*System.out.println("Logging before method execution");*

*}*

*@After("execution(\* Calculator.\*(..))")*

*public void logAfter() {*

*System.out.println("Logging after method execution");*

*}*

*}*

*AOP uses in this code:*

* AOP separates logging concern from Calculator business logic.
* AOP injects logging functionality into Calculator methods without altering their code.
* Log-related code is modularized into the LoggingAspect class.
* Makes it easier to manage and update logging behavior separately from Calculator operations.

NOTE:

Here the logging logic is encapsulated within the LoggingAspect class, separating it from the Calculator class. We use a pointcut expression (@Before("execution(\*

Calculator.\*(..))")) to specify where the advice (logging) should be applied. In this case, it applies to all methods of the Calculator class.

**Example 2:**

*import org.aspectj.lang.annotation.\*;*

*import org.springframework.security.core.context.SecurityContextHolder;*

*@Aspect*

*@Component*

*public class SecurityAspect {*

*@Before("@annotation(Secured)")*

*public void checkAuthorization() {*

*if (!SecurityContextHolder.getContext().getAuthentication().isAuthenticated()) {*

*throw new SecurityException("Access denied. User not authenticated.");*

*}*

*// Additional authorization checks can be added here*

*}*

*}*

* The checkAuthorization() method is annotated with @Before and targets methods annotated with @Secured.
* Before the execution of any method annotated with @Secured, it checks if the user is authenticated. If not, it throws a SecurityException.